Model-based validation of CANopen systems

Alexios Lekidis, Marius Bozga, Saddek Bensalem

Model-based validation of CANopen systems *

Alexios Lekidis, Marius Bozga, Saddek Bensalem
Univ. Grenoble Alpes, VERIMAG, F-38000 Grenoble, France
CNRS, VERIMAG, F-38000 Grenoble, France
firstname.lastname@imag.fr

Abstract

CANopen is an increasingly popular protocol for the design of networked embedded systems. Nonetheless, the large variety of communication and network management functionalities supported in CANopen can increase significantly systems complexity and in turn, the needs for system validation at design time. We present hereafter a rigorous method based on formal modeling and verification techniques, allowing to provide a comprehensive analysis of CANopen systems. Our method uses BIP, a formal framework for modeling, analysis and implementation of real-time, heterogeneous, component-based systems and the associated BIP tools for simulation, performance evaluation and statistical model-checking.

1 Introduction

Fieldbus protocols provide efficient solutions to important issues occurring in embedded system design nowadays. Such issues include managing system complexity, reducing communication cost as well as providing guarantees for functional and real-time requirements. A high-level protocol included in this family is CANopen [1]. This protocol is getting increasing popularity thanks to a vast variety of communication mechanisms, such as time or event-driven, synchronous or asynchronous as well as to additional support for time synchronization and network management. Moreover, it provides a high-degree of configuration flexibility, requires limited resources and has therefore been deployed on many existing embedded devices.

Applications using CANopen as their communication protocol can be found in automotive systems. In this domain, it is used as a high-level protocol on top of Controller Area Network (CAN) [2], in order to organize and abstract its low-level communication complexity. Since it offers parametrization according to predefined standards and manufacturer-specific device specifications, CANopen has also been deployed in distributed control systems, maritime electronics, medical devices, railway applications, photovoltaic and building automation systems e.t.c.

To the best of our knowledge, the existing tools for simulation, analysis and validation of CANopen systems are very limited. Vector GmbH [1] provides a powerful tool for the simulation of such systems, the CANalyzer [3]. It also contains a CANopen extension, namely the CANalyzer.CANopen. A relevant tool can be found in youCAN stack prototypes [4], provided by port GmbH [2]. CANopen Magic [5] is an interactive tool from Embedded Systems Academy [3], providing an interface for the development and simulation of applications using the protocol. Nonetheless, these tools are not able to perform timing analysis and validation. Furthermore, their use in the design of correct, functional CANopen systems requires high expertise. Likewise, they are targeting an industrial use and therefore their evaluation versions can only be used to familiarize with the protocol. Subsequently, they have limitations on the network size and the protocol functionalities. On the other hand, the equally powerful tools for CAN, capable of performing both timing analysis and performance evaluation, such as RTaW-Sim [6], are not implementing the CANopen protocol.

The aforementioned considerations are present, because CANopen is a fairly complex protocol and the interactions between the different types of communication mechanisms are very subtle. Thus, the protocol primitives can be easily misused, leading to poor, non-functional systems. A particular example is that even though it offers a wide range of services and communication mechanisms, the proper use of them is left to the device manufacturer as well as the application developer. The default method of setting the protocol parameters does not apply in many cases. An efficient solution to this issue is the availability of validation support at design time. Previous studies [7] [8] have illustrated that conformance testing can be used as a validation method in CANopen systems, due to its capability of verifying system integrity as well as the protocol's error-free functionality. However, the lack of functional error detection and performance analysis in earlier stages of the development cycle is a considerable design

*The research leading to these results has been partially funded by the French BGLE project ACOSE

[1] https://vector.com/
[2] http://www.port.de/
[3] http://www.esacademy.com/

limitation for such systems.

In this work we present an alternative validation method based on the use of formal modeling and verification techniques. We provide a systematic way to construct models of CANopen systems using the BIP component framework [9]. These models are constructed systematically, using a structural translation principle, from the protocol's entities and communication mechanisms. We show that the models obtained are faithfully compliant with the protocol's functional and timing aspects and can be used for either functional verification or performance analysis, using existing simulation and statistical model-checking tools available for BIP. As far knowledge is concerned, this method is the first attempt to obtain formal models for the CANopen protocol.

The rest of this paper is organized as follows. Section 2 provides a brief introduction to CANopen. Section 3 presents briefly the BIP framework along with its associated tools and discusses the techniques used for verification. Section 4 introduces the modeling and structural translation principles of CANopen systems in BIP. Section 5 provides experimental results of the model-based structural translation on existing benchmark systems and discusses current validation issues. Finally, Section 6 provides conclusions and perspectives for future work.

2 Overview of CANopen

CANopen is based on a master/slave architecture for management services, but concurrently uses the client/server communication model for configuration services as well as the producer/consumer model for real-time communication services. A comprehensive introduction to the protocol can be found in [10]. Unlike other Fieldbus protocols it does not require a single master controlling all the network communication. A CANopen system is specified by a set of devices (Figure 1), which in turn use a set of profiles, in order to define the device-specific functionality along with all the supported communication mechanisms. The communication profile defines all the services that can be used for communication and the device profile how the device-specific functionality is made accessible. The communication profile is defined in the DS-301 standard [1], whereas the device profiles providing a detailed description on CANopen's usage for a particular application-domain, are defined in the DS-4xx standards [4]. If CANopen systems require configurations or data access mechanisms not covered by the standard communication profile, profile extensions can also be defined. These are called Frameworks and are found in the DS-3xx standards [4].

The protocol's communication mechanisms according to the DS-301 are specified by standard *Communication Objects (COB)*. All the COBs have their own priority and are transmitted through regular frames of the chosen

[4]http://www.can-cia.org/index.php?id=440

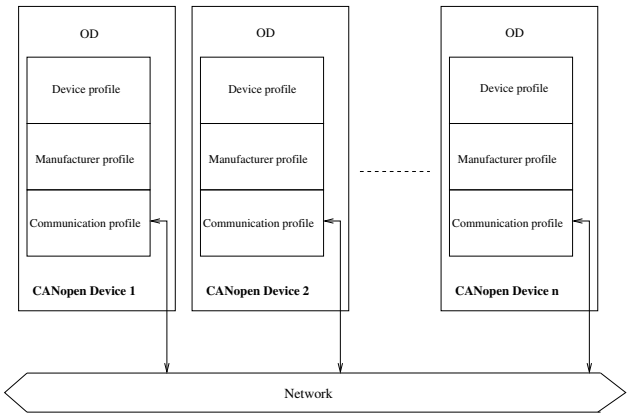

Figure 1: Communication in a CANopen system

lower-layer protocol. They are generally divided in the following main categories:

- *Network Management objects (NMT)*, used for the initialization, configuration and supervision of the network
- *Process Data Object (PDO)*, used for real-time critical data exchange
- *Service Data Object (SDO)*, used for service/configuration data exchange
- *Predefined objects*, found in specific entries in every OD. The featured objects in this category are:
  - *Synchronization object (SYNC)*, broadcasted periodically to offer synchronized communication as well as coordinate operations
  - *Timestamp object (TIME)*, broadcasted asynchronously to provide accurate clock synchronization using a common time reference
  - *Emergency object (EMCY)*, triggering interrupt-type notifications whenever device errors are detected

All the aforementioned objects are stored in a centralized repository, called Object Dictionary (OD), which holds all network-accessible data and is unique for every device. Commonly used to describe the behavior of a device, it supports up to 65536 objects. The COBs are spread to distinct areas, defining communication, device and manufacturer specific parameters. The latter are left empty to be used by manufacturers, in order to provide their own device functionalities.

The following sections provide a brief introduction to the aforementioned COB categories. Detailed description of their use along with sample network configuration parameters can be found in [11].

2.1 Process Data Objects (PDO)

The real-time data-oriented communication follows the producer/consumer model. It is used for the transmission of small amount of time critical data. PDOs can transfer up to 8 bytes (64 bits) of data per frame and are divided in two types: The transmit PDO (TPDO) denoting data transmission and the receive PDO (RPDO) denoting data reception. Therefore, a TPDO transmitted from a CANopen device is received as an RPDO in another device (Figure 2). Additionally, the supported scheduling modes are:

- *Event driven*, where the transmission is asynchronous and triggered by the occurrence of an object-specific event
- *Time driven*, where transmission is triggered periodically by an elapsed timer
- *Synchronous transmission*, triggered by the reception of the SYNC object, further divided in:
  - Periodic transmission within an OD-defined window (synchronous window), termed as *Cyclic PDO* transmission
  - Aperiodic transmission according to an application specific event, termed as *Acyclic PDO* transmission
- *Individual polling*, triggered by the reception of a remote request (see [12])

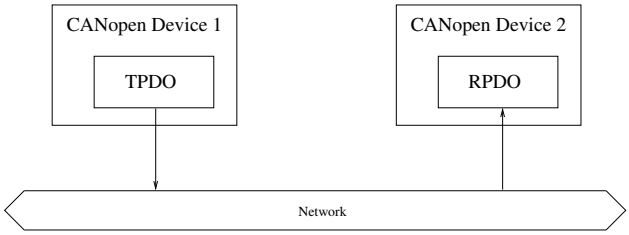

Figure 2: PDO communication

Each PDO is described by two OD entries: The Communication Parameter and Mapping Parameter. For a TPDO the former indicates the way it is transmitted in the network and the latter the location of the OD entry or entries combined, in order to build the payload. On the contrary for a RPDO the former indicates how it is received from the network and the latter the decoding of the received payload. The Communication Parameter entry includes the *Communication Identifier (COB-ID)* of the specific PDO, the scheduling method, termed as *transmission type*, the *inhibit time* and the *event timer*. The inhibit time defines the shortest and the event timer the longest time duration between two consecutive transmissions of the same PDO. The Mapping Parameter can be re-configured statically or dynamically through an SDO.

2.2 Service Data Objects (SDO)

The service oriented communication follows the client/server model. It supports large, non-critical data transfers and uses three modes to allow peer-to-peer asynchronous communication through the use of virtual channels:

- *Expedited transfer*, where service data up to 4 bytes are transmitted in a single request/response pair.
- *Segmented transfer*, where service data are transmitted in a variable number of 8-byte request/response pairs, termed as segments. The first pair is termed as initiation request or response respectively. In particular it consists of an initiation request/response followed by 8-byte request-response segments.
- *Block transfer*, optionally used for the transmission of large amounts of data as a sequence of blocks, where each one contains up to 127 segments.

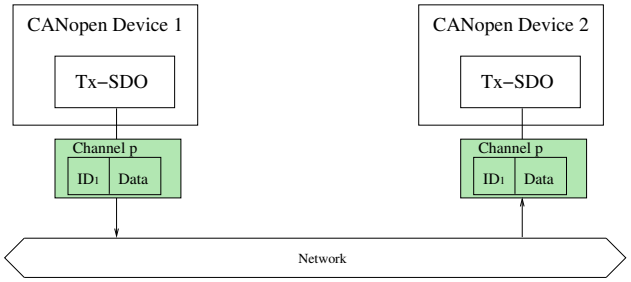

Figure 3: SDO communication

A CANopen device can either receive or request an SDO, therefore these objects are not separated as the PDOs, instead they are distinguished by two frames: the transmit SDO (Tx-SDO) and the receive SDO (Rx-SDO). The communication is always initiated by a device defined as client in the network towards the server, nonetheless information is exchanged bidirectionally with two services: *Download* and *Upload*. The former is used when the client is attempting service data transmission to the server, whereas the latter when it is requesting data from the server. In both services the use of the virtual peer-to-peer channel (Figure 3) ensures that a Tx-SDO request is acknowledged by an Rx-SDO response. For this reason service data transmission is regarded as the only confirmed communication mechanism of CANopen. Virtual channels can be configured dynamically by the Master device. If transmission errors are detected during the communication either on the client or the server side, data transfer is aborted through the SDO abort frame. SDOs are used for configuration and parametrization, but also allow the transmission of a large quantity of asynchronous data, consequently they are always assigned a lower priority than PDOs.

2.3 Predefined objects

These specific objects provide additional functionalities to the protocol. Their transmission is following the producer/consumer communication model. Particularly, the SYNC and the TIME object are always transmitted from a specific device (static OD configuration), whereas the EMCY object can be transmitted by any device in the network (dynamic OD configuration). The Predefined objects are always assigned with a high priority, in order to be transmitted as soon as possible.

3 The BIP component framework

The BIP framework (Behavior-Interaction-Priority) [9] supports a layered component construction methodology, facilitating the hierarchical system composition. The lower layer (Behavior) is described by finite-state automata or Petri-Nets and models the behavior of transition systems, termed as atomic components. Each transition is labeled by an action name, termed as port, but also associated with a guard and functions manipulating a set of variables. Guards are Boolean expressions enabling conditions in the component states. The use of ports in the second layer (Interaction) defines strong or loose synchronization upon data exchange, through the use of connectors. A connector is a list of ports of atomic components which may interact. Thus, an interaction is defined as strong synchronization, when all the ports of a connector are involved (graphically represented by a bullet), whereas in the opposite case it is defined as loose (graphically represented by a triangle). The third layer (Priority) restricts any non-determinism between simultaneously enabled interactions. A set of atomic components can be composed into a generic compound component by the successive application of connectors and priorities.

Figure 4 illustrates an example of a BIP composite component, comprised by two atomic components, the Sender and the Receiver. The ports *TICK*, *SEND* and *RECV* are used for the interactions between them. Each time both components are in the *idle* state the interaction involving the *SEND* and *RECV* ports is enabled. Its selection will lead to an update of variable *r*. The Sender and Receiver will respectively move to the *transmit* and the *receive* state. Consequently, both components will interact through the port *TICK* and increment variable *t*. Nevertheless, the Receiver component is also able to interact through the *EXE* port, in order to receive interruptions from other components. This conflict is resolved deterministically by priority π_1 : *TICK* $<$ *EXE*, allowing the transition involving port *EXE* to be chosen, when they are both enabled. On the contrary, port *COM* of the Sender component is not enabled as long as variable *t* is less than a specific value (here 100), due to the specified guard. As an interrupt may trigger port *EXE* before this value is reached, port *TICK* is evenly enabled in the *idle* state of the Receiver component.

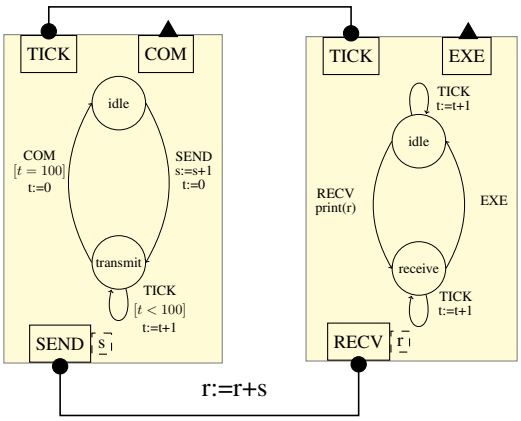

Figure 4: BIP components example

4 Modeling CANopen in BIP

CANopen systems in BIP consist of two communication layers. The top (application) layer components represent CANopen devices, responsible for frame transmission or reception, respectively called Device components. Following the CANopen specification, the model defines always a Device component as the Master, responsible for the transmission of the SYNC object and all the remaining Devices as Slaves. For the bottom (network) layer, we consider that communication is handled by the Controller Area Network (CAN) protocol. The CANopen Device component interacts with the CAN protocol, in order transmit or receive frames through the Bus. Therefore, prior to the derived structural translation of the CANopen primitives and communication mechanisms in BIP, we provide a brief introduction to the CAN protocol model in BIP, introduced in [13].

4.1 CAN protocol model

In our previous work [13] we developed a BIP model for the classic CAN as well as the newly developed CAN FD protocol [14]. The construction of the protocol model is using a library of CAN components, which in turn allows modularity and reusability. The soundness of the model was proved by the application in benchmark automotive systems, indicating similar results with RTaW-Sim [6].

The BIP model uses two generic types of components: the CAN station and the CAN bus. The former represents the hardware transceivers and the acceptance filters of the protocol and serves as an intermediary, in order to transmit frame requests generated from the upper layer to the Bus or equally deliver the received frames from the Bus to the upper layer. Thus, it is modeled as a compound component consisting of the Controller and the Filter atomic components accordingly. The latter represents the Bus functionality, preserving entirely its arbitration and broadcast mechanisms. Data transmission is synchronous, that is, all stations receive synchronously the frames sent by any of them. Furthermore, the underlying communica-

tion is a two-step process: first data are transmitted to the CAN bus and consequently broadcasted to all the CAN stations, including the sender. The transmission of each CAN frame field is followed by strong synchronization between the CAN stations and the CAN bus, through the use of interactions between the ports.

The CAN protocol model architecture is presented in Figure 5. It uses two groups of ports for its interactions, consisting of:

1. The *REQUEST* port (frame transmission), the *RECV* port (frame reception) and the TICK port used for the interactions with the upper layer.
2. The *SOF*, *ARBITRATION*, *CONTROL*, *DATA*, *ACK*, *EOF* ports used for the interactions between the CAN station and the CAN bus component

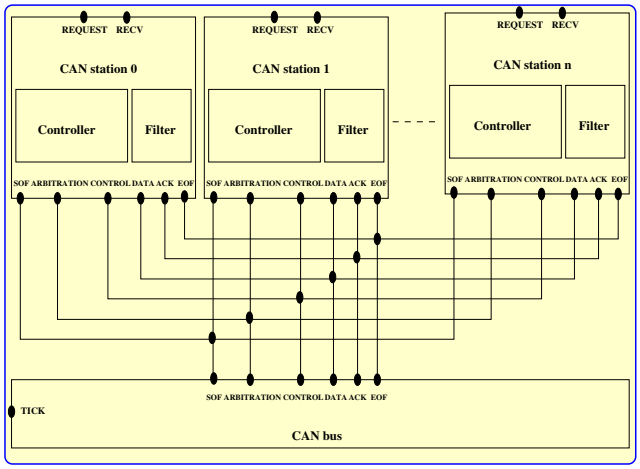

Figure 5: Generic model of a CAN/CAN FD system

The time needed for the transmission of each frame field is fixed. It is modeled as a discrete time step advance and denoted by the port *TICK*. However, the overall frame transmission time will vary, due to the blocking time and to the number of additional bits added as a result of bit-stuffing. If the frame payload is known beforehand, this number is calculated directly from the sequence of transmitted bits, whereas in the opposite case it can be rather chosen from a probabilistic distribution provided as an input to the model. Additionally, the blocking time will depend on the choice of the queuing policy for each CAN station component, and on the selection or not of transmission offsets as well as of abortable or non-abortable transmission requests [15]. The considered types of queuing policy in the model are: First-In-First-Out (FIFO) or High Priority First (HPF), where the selection is based on application-specific criteria.

4.2 CANopen model

The modeling of CANopen systems in BIP is structural. Every Device component is composed from several subcomponents, corresponding to COBs present in the device OD. As illustrated in Figure 6, the generic CANopen Device component is composed of three parts: a transmitting part (TRANSMIT), a receiving part (RECEIVE) and a third part involving both transmission and reception (TR). Each part consists of a set of components, implementing the protocol's communication mechanisms. Each atomic component is directly derived from a COB of the device OD, such that it will belong to one of the main categories mentioned in Section 2. In particular, PDO components can either exist exclusively only in the TRANSMIT or the RECEIVE part, or they can also be unused for the specific Device, meaning that they will not exist in any part. The same policy applies to SDO components with the difference that if they exist for the specific Device, they are included in the TR part. Furthermore, only one of the dashed SDO components is allowed to operate in the system at a time, thus the interactions between them are not maximal (loose synchronization). In the Predefined objects component category though only one Device can exist in the transmitting part and all the other on the receiving, meaning that they are exclusive for every Device. Therefore, only one of the dashed SYNC objects will be associated with the Device of Figure 6.

Each component is responsible for the handling of a COB as a frame and consists of the tuple: *(id,length,payload)*, where *id* is the value of the COB-ID for a particular frame. In the model it belongs to the protocol's default allocation scheme, termed as Predefined Connection Set [1] augmented by the identifier of the transmitting node. Thereafter, *length* contains the length of data and *payload* the actual data of the frame.

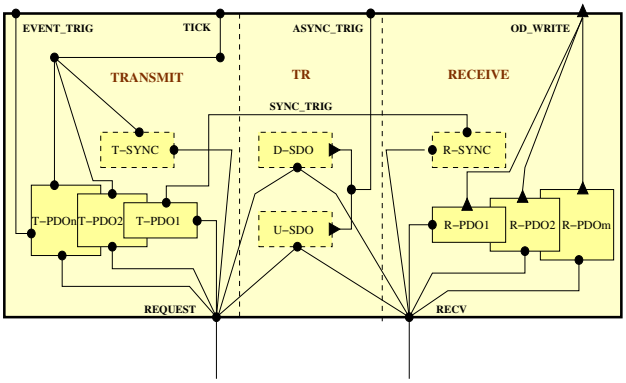

Figure 6: Generic CANopen Device component

We accordingly detail the behavior of the generic components used in the model, according to the COB category they belong. Each component, except the ones belonging to the SDO category, is atomic and described by abbreviations. These denote the part it belongs and the name of the object derived from, i.e SYNC Transmitter (T-SYNC) or SYNC Receiver (R-SYNC).

The generic CANopen Device component consists of four groups of ports using strong or loose synchronization upon interactions:

- The first implements interactions between different

CANopen objects, such as the *SYNC_TRIG* port

- The second implements interactions between the Device component and the lower communication layer, such as the *REQUEST*, *RECV* ports
- The third implements interactions between the Device component and application-specific components, such as the *EVENT_TRIG*, *ASYNC_TRIG*, *OD_WRITE* ports
- The fourth implements specific interactions for general synchronization and invokes all the previously listed groups, such as the *TICK* port

4.2.1 Process Data Objects (PDO)

The PDO component types implement all the supported scheduling policies, as illustrated in Section 2.1. Consequently they can be of three types: SYNC-triggered, time-triggered and event-triggered. Each type is further divided in two categories: T-PDO and R-PDO.

Each T-PDO component is responsible for the correct initialization and generation a TPDO (*REQUEST* port). In particular, the SYNC-triggered T-PDO component following the interaction between its *SYNC_TRIG* port and the R-SYNC component (Section 4.2.3), generates a synchronous PDO or performs another device-specific action. Evenly triggered by external interrupts is the event-triggered T-PDO component, through the port *EVENT_TRIG*. Finally, the time-triggered component implements a specific timer modeling the time step advance, through the *TICK* port. When this timer expires a time driven PDO is generated.

The corresponding RPDO components are responsible for the reception of a specific COB frame, provided as a parameter. They are triggered by lower-layer frame receptions (*RECV* port) and subsequently check the id of the received frame. If it is the expected frame its payload is written to the OD of the receiving Device component, through the port *OD_WRITE*. The particular OD entry is provided by the Mapping Parameter corresponding to the specified COB. This process may accordingly trigger a device-specific action.

4.2.2 Service Data Objects (SDO)

The SDO components are of two types: SDO Download (D-SDO) and SDO Upload (U-SDO) according to the protocol's communication mechanisms. The D-SDO and U-SDO components are responsible for configuration data exchange in the model, using one of the mechanisms presented in Section 2.2. They correspond respectively to the SDO Download mechanism and the SDO Upload mechanism. The Device transmitting the actual data is associated with the Tx-SDO COB-ID, whereas the Device receiving them with the Rx-SDO COB-ID. The D-SDO and U-SDO are implemented as compound components in the model, consisting of a Client and a Server atomic component. The SDO components do not implement any timing model, since service data transmission in CANopen is asynchronous. The Client component is always initiating data transmission, following the reception of an external event, through the *ASYNC_TRIG* port. The D-SDO Client component is presented in Figure 7. Apart from the *ASYNC_TRIG* port it interacts with the *REQUEST* and *RECV* ports, used for interactions with the lower communication layer. All its remaining ports are internal. Initially in the *S1* state, it moves to the *S2* whenever it is triggered by an asynchronous event. Accordingly, it determines if service data transmission is expedited or segmented. After the data request (*REQUEST* port) it remains in the *S3* state, until it receives (*RECV* port) a frame whose id is *1408+clientID* and the received server command specifier (scs) is valid. *ClientID* is the identifier of the specific client device. If the transmission was expedited (bit *e* from payload byte 0 is set) it will return to the initial state (S1), otherwise it will repeat the aforementioned process for all the subsequent segments, initialized according to the device OD and denoted in the model by variable *N* (model parameter). The variable counter is decremented in every successful transmission of a request/receive pair, until it is equal to 1, indicating the last segment (bit *c* from payload byte 0 is set). Afterwards, the component moves to the initial state, otherwise it proceeds to the next segment by the transition *next_segment*. The *toggle* variable is used to identify the sequence of successfully received request/response segments (bit *t* from payload byte 0).

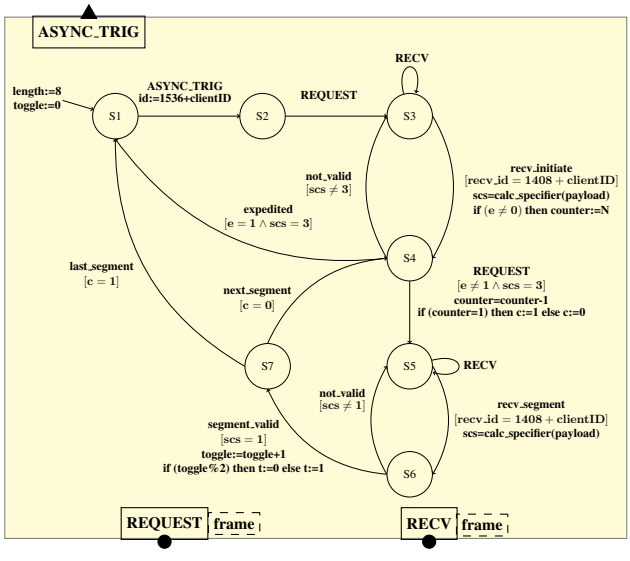

Figure 7: D-SDO Client component

4.2.3 Predefined objects

This category is focused on the SYNC object, as other objects are not considered mandatory (described in [11]).

As for PDO, the SYNC components are divided in two categories: T-SYNC and R-SYNC (Figure 8). The T-SYNC component is responsible for the SYNC frame transmission. It consists of the states *idle*, *transmit* and the ports: *TICK* and *REQUEST*. Initially it is in the *idle* state, where it interacts through the *TICK* port. This port denotes the notion of step time advance in the model, which is calculated and stored in the variable *t*. When *t* is equal to the value of the SYNC period (defined in the device OD) the transmission is triggered by an internal move to the state *transmit*. The transmitted *frame* is initialized with the SYNC object parameters before the transmission through the port *REQUEST*. Subsequently, the component moves to the trigger state. The R-SYNC component is controlling the SYNC-triggered PDO transmission. It only triggers a frame transmission upon the successful reception of the SYNC frame. This component consists of the states *idle*, *receive* and the ports: *SYNC_TRIG* and *RECV*. When a frame is received through the *RECV* port, used for the interactions with the lower communication layer, it will move to the *receive* state. It returns to the *idle* state either by triggering the transmission of a PDO frame (*SYNC_TRIG* port), or internally. The choice is controlled by a specific guard.

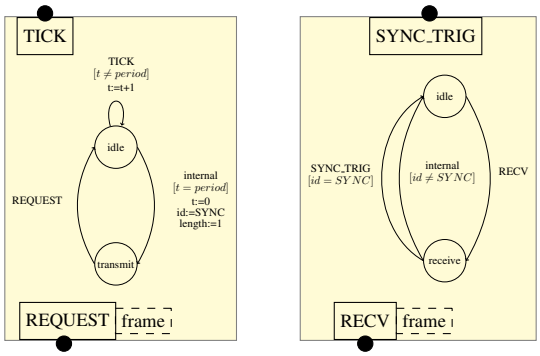

Figure 8: T-SYNC and R-SYNC components

4.2.4 Timing and version issues

A constraint that has to be carefully considered in our model is the choice of the time step advance for the *TICK* interaction. Its granularity has to be relative with the baud-rate (speed) of the CAN protocol. Therefore we consider the time needed for the transmission of one bit to the Bus equal to one-step advance in our model. For example a baud-rate of 500 kbit/s, corresponds to a time step advance of 2 microseconds (μs). Subsequently, $2\mu s$ of real time will be taken as a one-step advance in our model.

The version of the CANopen protocol model represents the functionality of the most recent communication profile [1], which additionally implies that the SYNC object is not anymore mapped to an empty frame, but includes an 1-byte counter as payload. Moreover, currently we don't consider hardware or transmission errors. Therefore, the SDO abort frame is not included in the model.

4.2.5 Concluding remarks on the modeling

The construction of a formal model facilitated the identification of some important issues in CANopen communication. Initially, in SDO communication the data size parameter is optional and usually not indicated in CANopen systems before as well as during the transfer. Even though this type of objects should always be addressed with the lowest priority, the receiver cannot perform a consistency check, which is consequently reducing the robustness of the protocol. Another important issue is related to the number of unused data bytes in some SDO frames, instead filled with padding, in order to follow the 7-byte data request/receive pair specification. The outcome is the introduction of significant overhead to the lower-layer transmission protocol, which might cause additional delays in the transmission of high-priority frames, especially during SDO block transfers.

Overall, the built component libraries contained 14 types of atomic components for CANopen and 3 types of atomic components for CAN/CAN FD.

5 Case study: Pixel Detector Control System

The conducted experiments focused on the Pixel Detector Control System (DCS), used as the innermost part for the ongoing ATLAS experiment at CERN's Large Hadron Collider (LHC) particle accelerator. For the particular case study we consider an extension to the test beam of 2002, previously presented in [16], used for the calibration and performance evaluation of the detector modules used in the experiment.

The chosen test beam is presented in Figure 9 and consists of two Detector systems, where each one contains four pixel detector modules. Each pixel detector module is equipped with a temperature sensor, used in order to measure its operating temperature and accordingly determine its lifetime. The measurements are subsequently provided as input to a thermal interlock system (*Interlock Box*) and a plug-on I/O board manufactured in CERN, named as *ELMB* (Embedded Local Monitor Board), in order to be transmitted to a Detector Control System (DCS) Station through the CAN Bus, using CANopen as the communication protocol. The application software as well as the hardware configuration for the ELMB board can be found in [17]. This manual also provides a full listing of the Object Dictionary, defining not only the standard objects according to the DS-401 Device Profile [18], but also manufacturer-specific objects for the ELMB.

A new scan cycle begins every 1 second and in the course of it all the pixel detector modules are scanned. A TPDO2 frame is transmitted whenever a change of a modules temperature value is detected. This change is particularly termed as *Change-of-State (CoS)*. The transmitted frame contains the ADC readout in counts (ADC resolution). However, after a power-up or a reset of the ELMB

the ADC voltage ranges need to be re-calibrated through a TPDO3 frame. This frame contains the input voltage in μV and is transmitted prior to the generation of a TPDO2 frame. Since each temperature sensor is exposed to safety risks, the Interlock Box is responsible of comparing the input data to a reference value (threshold) as well as for the generation of a logical signal, if the temperature is found higher. The output of every Interlock Box module is provided to a Logic Unit, which is also monitored by an ELMB module. This module is used to transmit the generated signal as a TPDO1 frame, informing the associated pixel detector that it is overheated, in order to enable its Cooling Box. The coolant flow inside each Cooling Box is set and controlled by an expedited SDO frame. Therefore, two additional ELMB modules are considered, each one obtaining coolant flow data from a Regulator module. Subsequently they establish a peer-to-peer communication channel with the corresponding ELMB of each Detector module and transmit the data through an SDO Download operation. Finally, although the DCS Station is mainly used for data logging, it is also responsible for the periodical transmission of the SYNC frame, informing the ELMB module of every Detector to abort the current scan cycle and accordingly start a new one.

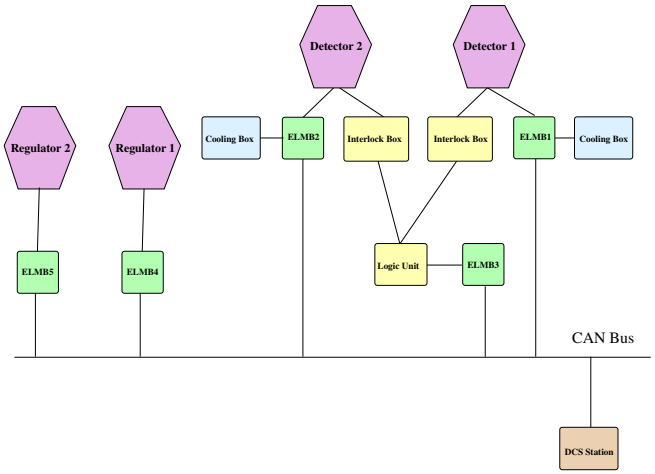

Figure 9: Pixel Detector Control System

The bit-rate of the CAN Bus for the particular test beam is set to 125kbit/s. An equally important remark is that, during the initialization phase of the system, the DCS Station initializes properly all the ELMB devices. This is achieved by an SDO Download operation storing all the COB-IDs correctly in their OD.

The existence of certain requirements for the DCS ensure the proper functionality of the system. They are divided in two categories: those concerning the physics and performance of the DCS individual subsystems, found in the CERN Document Server [5], and those related to the communication through CANopen. Specific requirements belonging to the second category are:

[5]http://cds.cern.ch/record/391176

1. In case of an increased sensor temperature the Logic Unit must inform the DCS Station rapidly, through a TPDO1 frame. Thus, the TPDO1 frame must have a zero blocking time, once triggered.

2. When ELBM1 or ELMB2 is reset a TPDO3 frame should be transmitted before a CoS in a pixel detector module is detected, since the generated TPDO2 will require an ADC conversion.

3. The coolant flow must be set at least once before a Cooling Box is required to cool an indicated pixel detector module. Consequently, ELMB4 and ELMB5 should initiate the transmission of the D-SDO frame before any other frame in the network is triggered.

We constructed the model of the Pixel Detector Control System, using the library of CANopen components presented in Section 4.2. The resulting BIP system for the DCS is illustrated in Figure 10. It is comprised by 39 atomic components forming the CANopen communication layer and 13 atomic components for the CAN protocol. The generated BIP model used 95 connectors (53 for the CANopen and 42 for the lower-layer communication model). The total number of transitions for this system was 427 (174 for the CANopen and 252 for the lower-layer communication model). Overall the model totals about 2300 lines of BIP textual code.

For the conducted experiments we used real temperature data provided as input to the model, thus deriving a distribution for the temperature changes, as well as the reference value (threshold) for the Interlock Box. Moreover, the external event triggering an SDO transmission was modeled as an asynchronous timer generating event interrupts, whenever it expired.

A real system time of 4 hours was simulated in 2 minutes and 43 seconds using the BIP simulator. The obtained results are illustrated in terms of minimum, average and worst-case frame response times in Figure 11. The COB-IDs of the existing frames in the system according to the Predefined Connection Set are represented in the horizontal axis. As it is observed the response times (in milliseconds) are highly dependent from the choice of lower-layer scheduling policy (here HPF). Due to the stochastic behavior of the system, the blocking time for each transmission varies according to the Bus load at the given instant. This variation between the minimum (zero blocking time) and the maximum (worst-case blocking time) depends on the frame identifier. In particular, the SYNC frame (COB-ID 128) has a relatively small variation compared to the D-SDO frames of ELMB1 and ELMB2 (COB-IDs 1540 and 1541 respectively). In this analysis the response time of the SDO frames is measured from the instantiation of the request frame until the transmission end of the response frame.

In order to evaluate the system requirements we describe the above requirements with stochastic temporal properties using the Probabilistic Bounded Linear Temporal Logic (PBLTL) formalism [19]. We accordingly

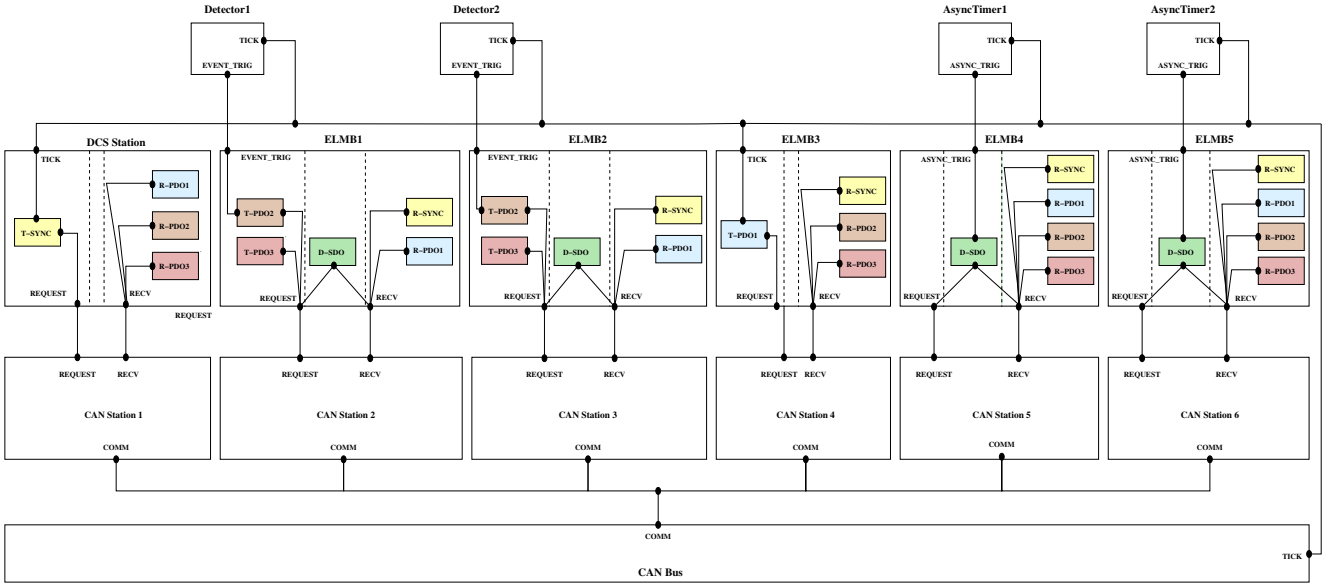

Figure 10: BIP model of the Pixel Detector Control System

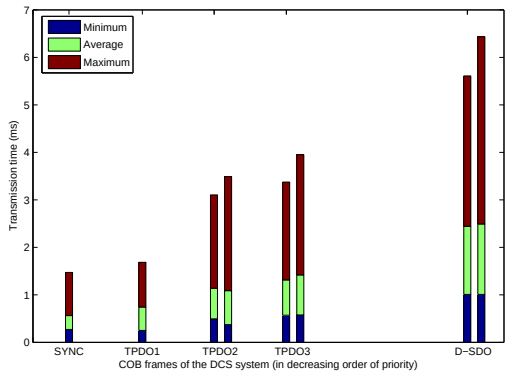

Figure 11: BIP frame response times

present the results derived properties after an extensive number of simulations using the SBIP model checker [6].

Property 1: Requirement 1. This property is expressed as $\phi_1 = \Box^{10000000}((T_{inhibit} - T_{TPDO1}) > 0)$, where 10000000 indicates the number of steps for each simulation, corresponding to a large number of communication cycles. Furthermore, $T_{inhibit}$ is the inhibit time and T_{TPDO1} the response time of the TPDO1 frame (COB-ID 388). For the DCS system $T_{inhibit}$ is equal to 1 sec, which much greater than the maximum response time of TPDO1 ($T_{TPDO1_{max}} = 1.72$ msec from Figure 11). Therefore $P(\phi_1) = 1$ and this requirement is always satisfied.

Property 2: Requirement 2. In the second experiment, we try to estimate the property $\phi_2 = \Diamond^{10000000}((T_{TPDO2} - T_{TPDO3}) < 0)$, where 10000000 is explained as above, T_{TPDO2} and T_{TPDO3} denote the response time of TPDO2 and TPDO3 following an ELMB reset. The conducted experiments have shown that if a scan cycle is initiated through the reception of the SYNC frame, a CoS can be detected before the generation of a TPDO3 frame. However, a reset in ELMB1 or ELMB2 occurred in approximately 3% of the simulations, thus this property was quantified as $P(\phi_2) = 0.005$. This probability is equal to the tool's level of confidence, thus the requirement is considered as satisfied.

Property 3: Requirement 3. Finally, we tested through simulation the property $\phi_3 = \Box^{24000}((t_{TPDO2} - t_{D-SDO}) > 0)$, where 24000 is the number of steps required for the initialization period as T_{init} is 2 sec, t_{TPDO2} is the system time at the end of the TPDO2 frame transmission and t_{D-SDO} is the system time at the beginning of the D-SDO frame transmission. Since the D-SDO frame was generated asynchronously, this property was quantified as $P(\phi_3) = 0.1$. As it is observed by Figure 12, focusing in a specific simulation, the TPDO2 frames from ELMB1 and ELBM2 finish their transmission before a D-SDO frame. Moreover the conducted experiments have shown that even when the D-SDO frame was generated before the first instance of a TPDO2 frame, it was mostly blocked due to its lowest priority for this system.

6 Conclusion and ongoing work

We have presented a systematic method to construct detailed functional models for CANopen systems in the BIP component framework. The construction method is fully structural, that is, it preserves the CANopen system structure in BIP, meaning systems consisting of a number of network-connected devices, which in turn, consist of a

[6] http://www-verimag.imag.fr/Statistical-Model-Checking.html

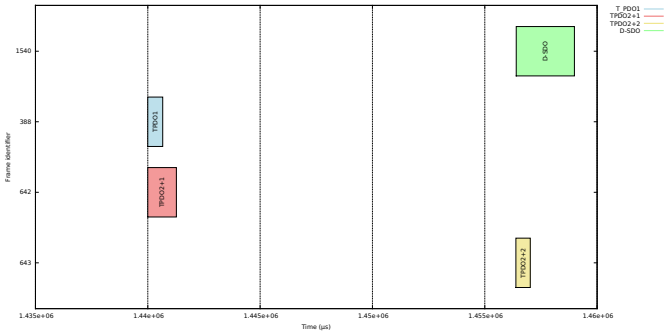

Figure 12: Response time graph for TPDO1, TPDO2 and D-SDO

number of interacting communication objects, as defined in the device and the communication profile accordingly. The model captures both functional and extra-functional aspects, referring to timing characteristics for periodic and aperiodic transmission. The models are fully operational, they can be tested, simulated and validated using statistical model checking tools available in the BIP toolset.

For the time being, our model uses the CAN protocol for the low-layer network communication. However, its use also raises practical limitations as with unconfirmed services, since there is no possibility of knowing when a frame is lost. A possible solution would be the use of individual polling from the Master device. Nevertheless, this method will produce additional overhead, if there is no CoS in a number of devices. Furthermore, the two COB-IDs defined for SDO communication allow only one client/server channel in the network at a time. In the opposite case collisions or conflicts are inevitable. Additionally, the growing use of extensive networks and the rising data load are increasing the complexity of CANopen systems nowadays. One of the main reasons behind this is the low bandwidth (1 Mbit/s) and the limitation in the network length (127 nodes and 25m max bus length). CAN FD was introduced, in order to ameliorate the former limitation, nonetheless the bandwidth is only increased during the data transmission period.

For all these reasons, we are working on further extensions, in order to support CANopen systems deployed on other wired or wireless protocols. The two most interesting protocols in this domain are the IEEE 802.3, used for wired Ethernet communication and the IEEE 802.11, used for wireless communication. In the scope of these extensions, we shall map the presented primitives and communication mechanisms of CANopen to each aforementioned protocol.

References

[1] CAN in Automation, "Application layer and communication profile, Draft Standard 301", February 2011.

[2] R. Bosch, "CAN specification version 2.0", *Robert Bosch GmbH, Stuttgart*, 1991.

[3] Vector Informatik GmbH, *CANanalyzer User Manual*, http://vector.com/vi_manuals_en.html.

[4] port GmbH, *youCAN CANopen prototyping*, http://www.port.de/fileadmin/user_upload/Dateien_IST_fuer_Migration/youCAN_e.pdf.

[5] Embedded Systems Academy, *CANopen Magic User Manual*, http://www.esacademy.org/products/getfile.php?filename=COMPDLLManual.pdf.

[6] N. Navet, A. Monot, J. Migge, et al., "Frame latency evaluation: when simulation and analysis alone are not enough", in *8th IEEE International Workshop on Factory Communication Systems (WFCS2010), Industry Day*, 2010.

[7] M. Barbosa, M. Farsi, C. Allen, and A. Carvalho, "Formal validation of the CANopen communication protocol", in *Fieldbus Systems and Their Applications 2003:(FET 2003): a Proceeedings Volume from the 5th IFAC International Conference, Aveiro, Portugal, 7-9 July 2003*, volume 5, July 2003, pp. 226–238. Elsevier Science Limited, IFAC.

[8] T. Schumann, "CANopen Conformance Test", in *Fieldbus Technology*, pp. 152–156. Springer, 1999.

[9] A. Basu, M. Bozga, and J. Sifakis, "Modeling heterogeneous real-time components in BIP", in *Software Engineering and Formal Methods, 2006. SEFM 2006. Fourth IEEE International Conference on*, 2006, pp. 3–12. IEEE.

[10] O. Pfeiffer, A. Ayre, and C. Keydel, *Embedded networking with CAN and CANopen*, Copperhill Media, 2008.

[11] Alexios Lekidis, Marius Bozga, Saddek Bensalem, "Rigorous Modeling and Validation of CANopen Systems", Technical Report TR-2014-1, Verimag Research Report, 2014.

[12] CAN in Automation, *Application Note 802*, August 2005.

[13] A. Lekidis, M. Bozga, D. Mauuary, and S. Bensalem, "A model-based design flow for CAN-based systems", in *14th International CAN Conference, Eurosites République, Paris*, 2013.

[14] R. Bosch, "CAN with Flexible Data-Rate specification", *Robert Bosch GmbH, Stuttgart*, 2012, http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can_fd_spec.pdf.

[15] D. A. Khan, R. I. Davis, and N. Navet, "Schedulability analysis of CAN with non-abortable transmission requests", in *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference*, 2011, pp. 1–8. IEEE.

[16] S. Kersten, K. Becks, M. Imhäuser, P. Kind, P. Mättig, and J. Schultes, "Towards a Detector Control System for the ATLAS Pixel Detector", 2002.

[17] Henk Boterenbrood, *CANopen application firmware for the ELMB*, November 2011, http://www.nikhef.nl/pub/departments/ct/po/html/ELMB128/ELMB24.pdf.

[18] CAN in Automation, "CANopen Device Profile for Generic I/O Modules, Draft Standard 401", June 2008.

[19] S. Bensalem, M. Bozga, B. Delahaye, C. Jegourel, A. Legay, and A. Nouri, "Statistical Model Checking QoS properties of Systems with SBIP", in *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, pp. 327–341. Springer, 2012.